



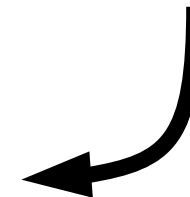
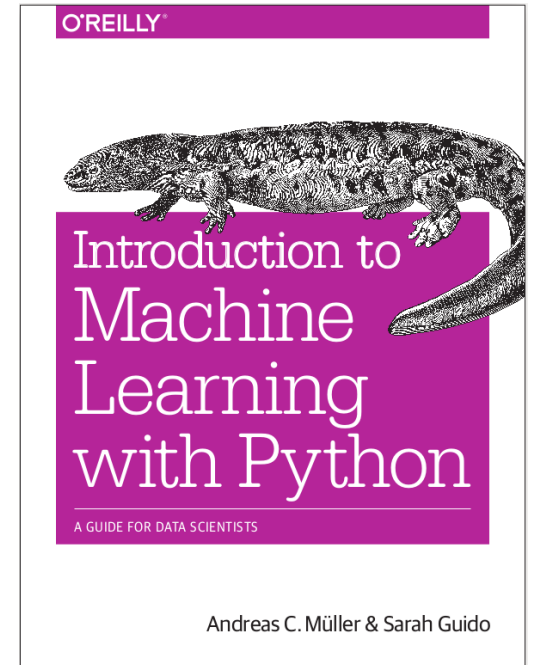
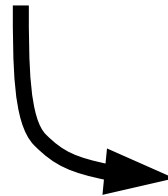
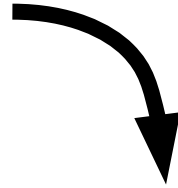
# Down to the metal with scikit-learn

Andreas Müller  
Columbia University,  
scikit-learn

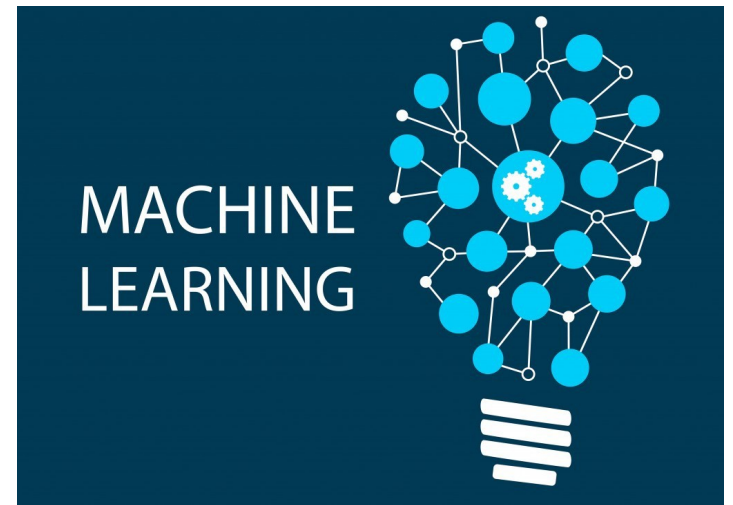


Alfred P. Sloan  
FOUNDATION





What is scikit-learn?



Classification  
Regression  
Clustering  
Semi-Supervised Learning  
Feature Selection  
Feature Extraction  
Manifold Learning  
Dimensionality Reduction  
Kernel Approximation  
Hyperparameter Optimization  
Evaluation Metrics  
Out-of-core learning

.....





Alexandre Gramfort

agramfort



Alexander Fabisch

AlexanderFabisch



Alexandre Passos

alextp



Andreas Mueller

amueller



Arnaud Joly

arjoly



Brian Holt

bdholt1



bthirion

bthirion



Chris Filo Gorgolewski

chrisfilo



David Cournapeau

cournape



Duchesnay

duchesnay



David Warde-Farley

dwf



Fabian Pedregosa

fabianp



Gael Varoquaux

GaelVaroquaux



Gilles Louppe

glouppe



Jake Vanderplas

jakevdp



Jaques Grobler

jaquesgrobler



Jan Hendrik Metzen

jmetzen



Jacob Schreiber

jmschrei



Joel Nothman

jnothman



Kyle Kastner

kastnerkyle



Lars

larsmans



Loïc Estève

lesteve



Shiqiao Du

lucidfrontier45



Mathieu Blondel

mblonde1



Manoj Kumar

MechCoder



Noel Dawe

ndawe



Nelle Varoquaux

NelleV



Olivier Grisel

ogrisel



Paolo Losi

paolo-losi



Peter Prettenhofer

pprett



(Venkat) Raghav (Rajagopalan)

raghavrv



Robert Layton

robertlayton



Ron Weiss

ronw



Satrajit Ghosh

satra



sklearn-ci



sklearn-wheels



Tom Dupré la Tour

TomDLT



Vlad Niculae

vene



Virgile Fritsch

VirgileFritsch



Vincent Michel

vmichel



Wei Li

weilinear



Yaroslav Halchenko

yarikoptic

# Mission

Commoditize and Democratize Machine Learning

# Basic API



# Representing Data



one sample

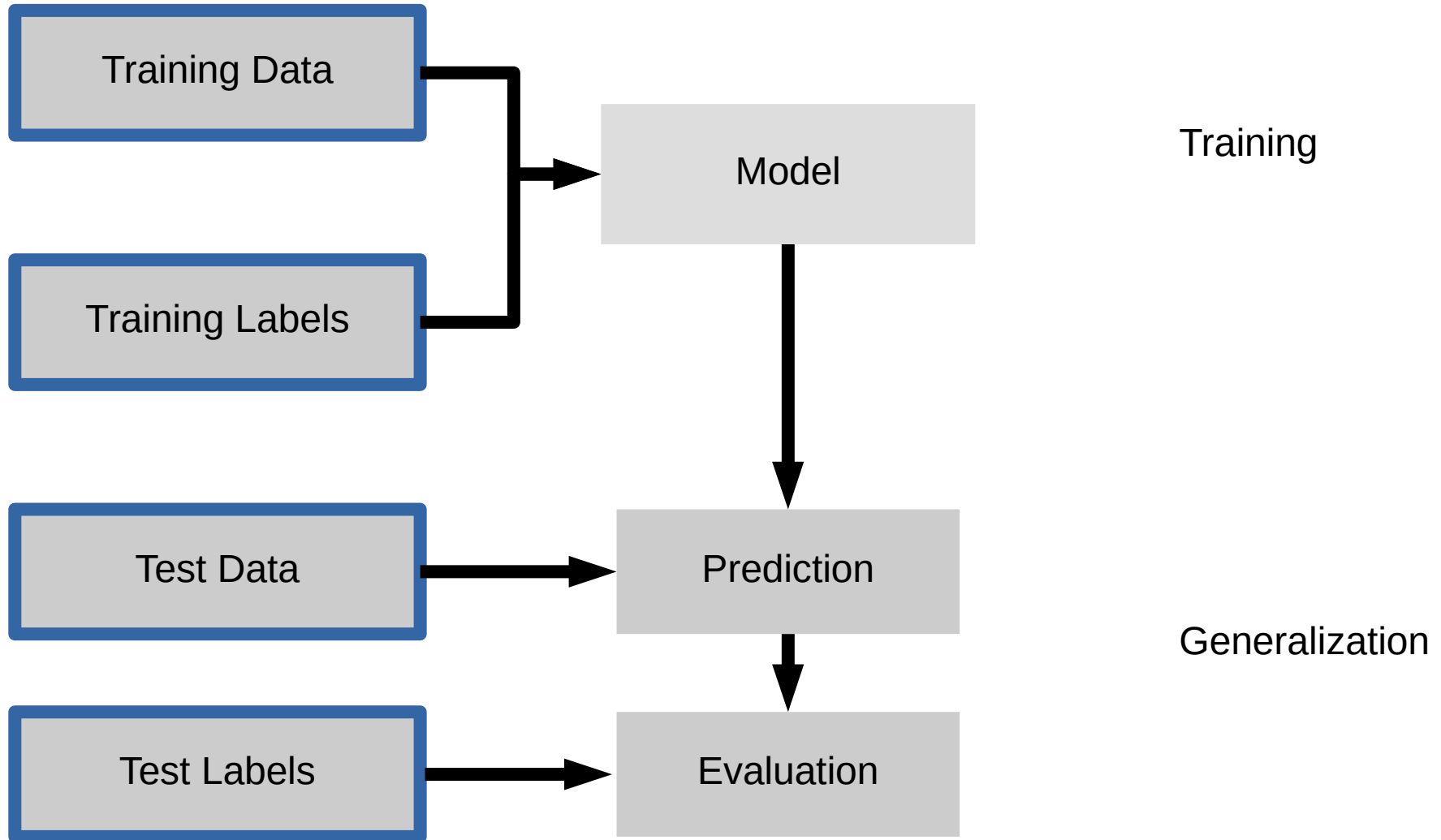
$$X = \begin{pmatrix} 1.1 & 2.2 & 3.4 & 5.6 & 1.0 \\ 6.7 & 0.5 & 0.4 & 2.6 & 1.6 \\ 2.4 & 9.3 & 7.3 & 6.4 & 2.8 \\ 1.5 & 0.0 & 4.3 & 8.3 & 3.4 \\ 0.5 & 3.5 & 8.1 & 3.6 & 4.6 \\ 5.1 & 9.7 & 3.5 & 7.9 & 5.1 \\ 3.7 & 7.8 & 2.6 & 3.2 & 6.3 \end{pmatrix}$$

one feature

$$y = \begin{pmatrix} 1.6 \\ 2.7 \\ 4.4 \\ 0.5 \\ 0.2 \\ 5.6 \\ 6.7 \end{pmatrix}$$

outputs / labels

# Supervised Machine Learning

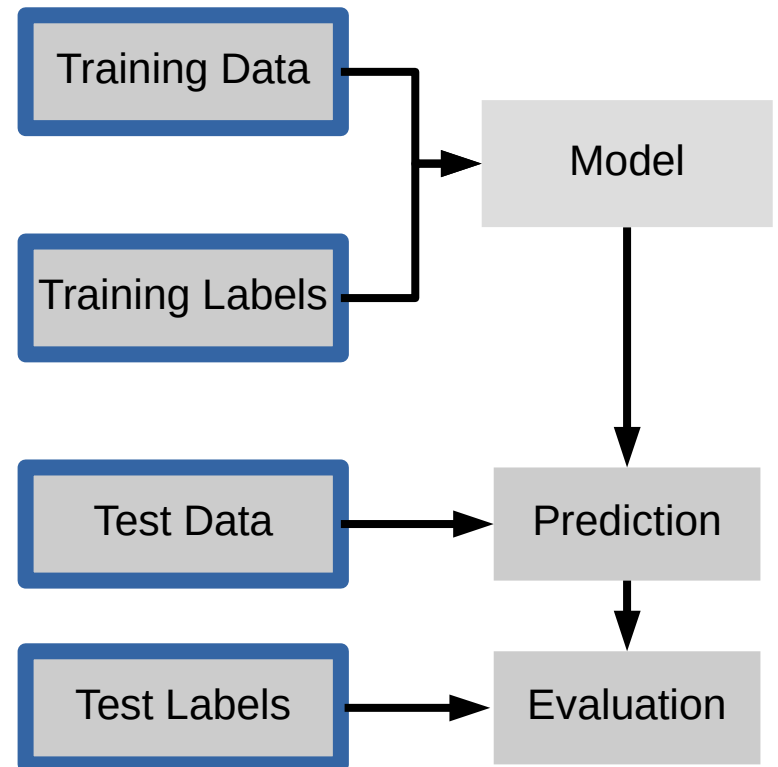


```
clf = RandomForestClassifier()
```

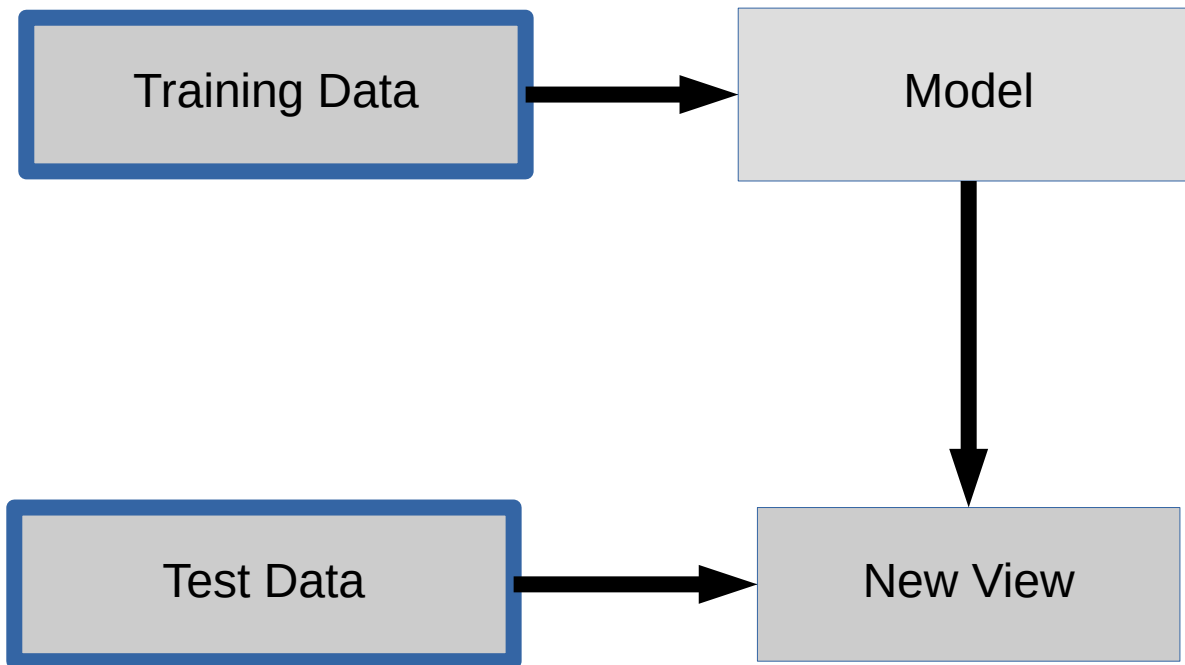
```
clf.fit(X_train, y_train)
```

```
y_pred = clf.predict(X_test)
```

```
clf.score(X_test, y_test)
```



# Unsupervised Machine Learning

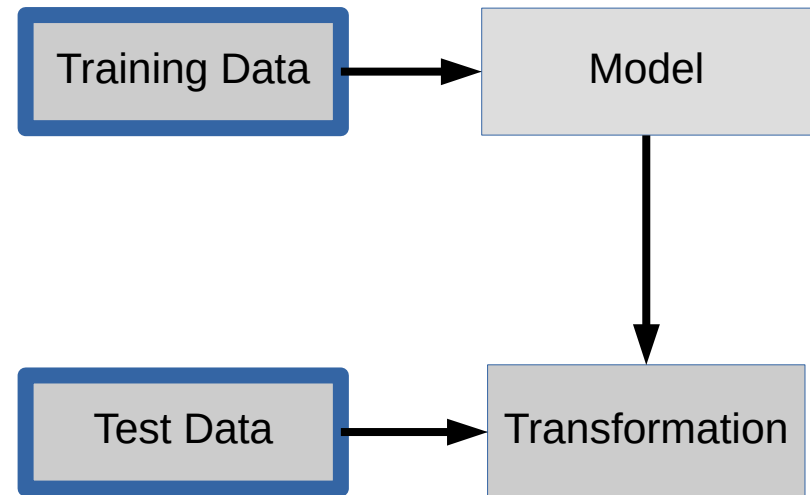


# Unsupervised Transformations

```
pca = PCA()
```

```
pca.fit(X_train)
```

```
X_new = pca.transform(X_test)
```



# Core API Summary

**`estimator.fit(X, [y])`**

**`estimator.predict`**

**`estimator.transform`**

---

Classification

Preprocessing

Regression

Dimensionality reduction

Clustering

Feature selection

Feature extraction

# Cross -Validated Grid Search

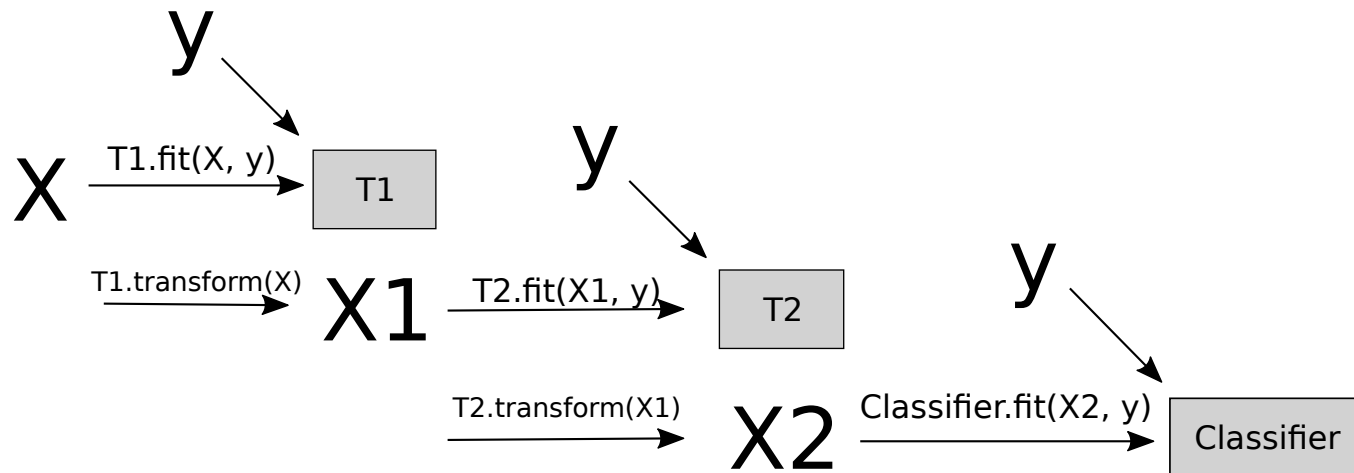
```
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y)
param_grid = {'C': 10. ** np.arange(-3, 3),
              'gamma': 10. ** np.arange(-3, 3)}
grid = GridSearchCV(SVC(), param_grid=param_grid)
grid.fit(X_train, y_train)
grid.predict(X_test)
grid.score(X_test, y_test)
```

# Pipelines

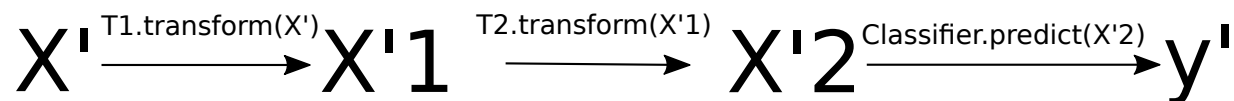
```
pipe = make_pipeline(T1(), T2(), Classifier())
```



```
pipe.fit(X, y)
```



```
pipe.predict(X')
```





# Pipelines

```
from sklearn.pipeline import make_pipeline  
pipe = make_pipeline(StandardScaler(), SVC())  
pipe.fit(X_train, y_train)  
pipe.predict(X_test)
```

# Guiding Ideas

Goals:

Maintainability  
Ease of use

Simple things should be simple,  
complex things should be possible.

Alan Kay





Programs should be written for people to read,  
and only incidentally for machines to execute.

Harold Abelson

# Simplicity

```
lr = LogisticRegression()  
lr.fit(X_train, y_train)  
lr.score(X_test, y_test)
```

# Consistency

```
grid = GridSearchCV(svm, param_grid)
grid.fit(X_train, y_train)
grid.score(X_test, y_test)
```

# Composition

```
feature_selection = RFECV(NaiveBayes())  
pipe = make_pipeline(feature_selection,  
                      RandomForestClassifier())  
grid = GridSearchCV(pipe, param_grid)
```



# Default Parameters

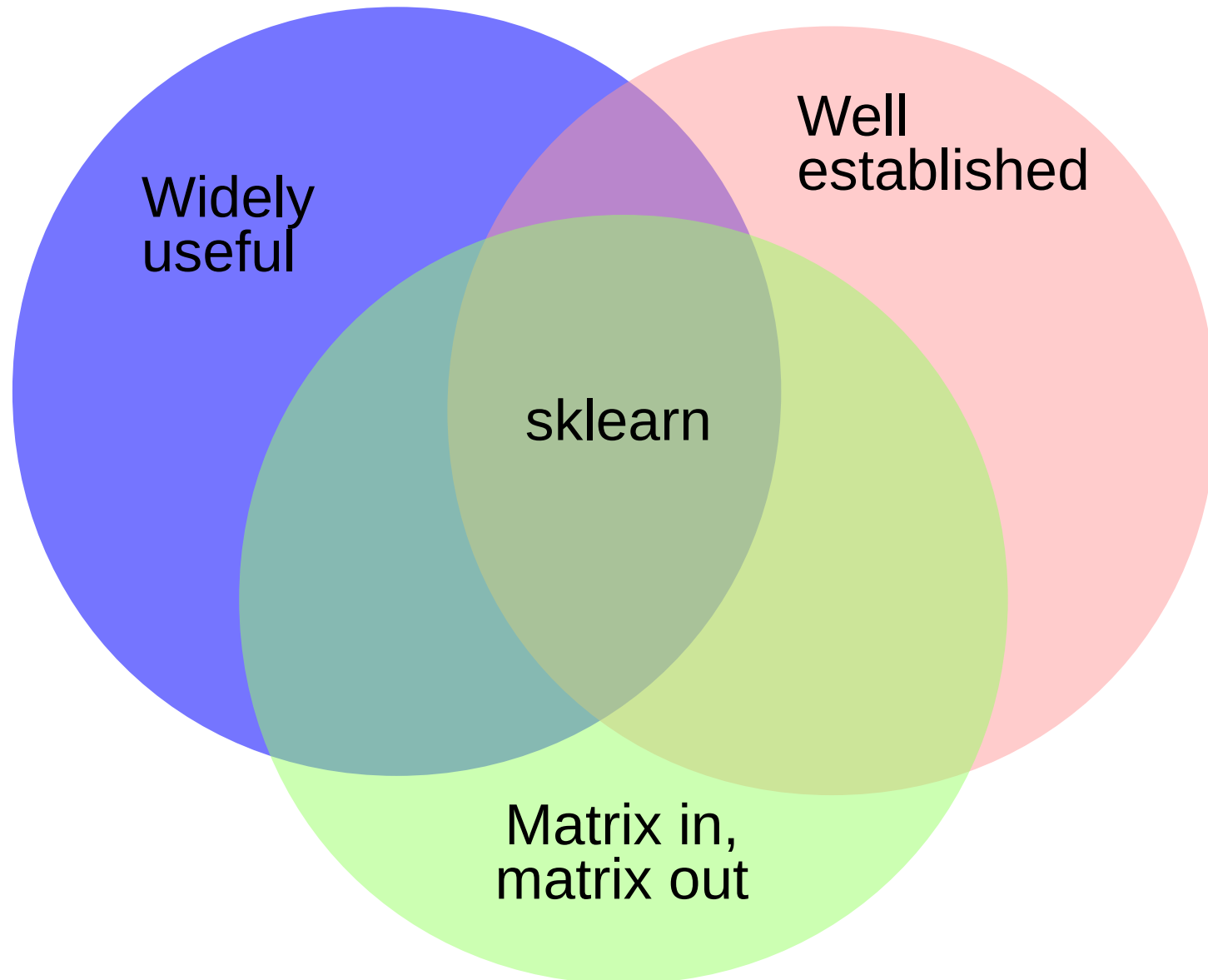
```
In [2]: clf = SVC()  
clf.fit(X_train, y_train)
```

```
SVC(self, C=1.0, kernel='rbf', degree=3, gamma=0.0, coef0=0.0,  
shrinking=True, probability=False, tol=0.001, cache_size=200,  
class_weight=None, verbose=False, max_iter=-1, random_state=None)
```

# Flat Class Hierarchy, Few Types

- Numpy arrays / sparse matrices
- Estimators
- Cross-validation objects
- Scorers

# Scoping



# Testing & Continuous Integration

Add more commits by pushing to the `more_repr` branch on `amueller/scikit-learn`.



**All checks have passed**

3 successful checks

[Hide all checks](#)



**ci/circleci** — Your tests passed on CircleCI

[Details](#)



**continuous-integration/appveyor/pr** — AppVeyor build succeeded

[Details](#)



**continuous-integration/travis-ci/pr** — The Travis CI build passed

[Details](#)



**This branch has no conflicts with the base branch**

Merging can be performed automatically.

**Squash and merge**



or view [command line instructions](#).

# Three way documentation

## 1.9. Ensemble methods

The goal of **ensemble methods** is to combine the predictions of several base estimators built with a given learning algorithm in order to improve generalizability / robustness over a single estimator.

Two families of ensemble methods are usually distinguished:

- In **averaging methods**, the driving principle is to build several estimators independently and then to average their predictions. On average, the combined estimator is usually better than any of the single base estimator because its variance is reduced.

**Examples:** *Bagging methods, Forests of randomized trees, ...*

- By contrast, in **boosting methods**, base estimators are built sequentially and one tries to reduce the bias of the combined estimator. The motivation is to combine several weak models to produce a powerful ensemble.

**Examples:** *AdaBoost, Gradient Tree Boosting, ...*

## `sklearn.ensemble.RandomForestClassifier`

```
class sklearn.ensemble. RandomForestClassifier (n_estimators=10, criterion='gini',
max_depth=None, min_samples_split=2, min_samples_leaf=1,
min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None,
bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0,
warm_start=False)
```

[\[source\]](#)

A random forest classifier.

A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and use averaging to improve the predictive accuracy and control over-fitting.

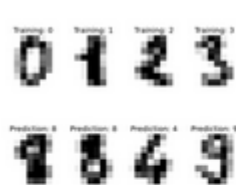
**Parameters:** `n_estimators` : integer, optional (default=10)

The number of trees in the forest.

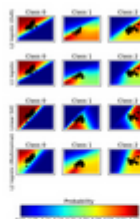
**criterion** : string, optional (default="gini")

The function to measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "entropy" for the information gain. Note: this parameter is tree-specific.

## Examples



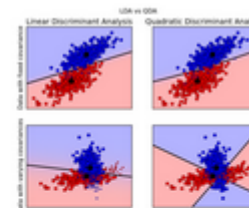
*Recognizing  
hand-written digits*



*Plot classification  
probability*



*Classifier comparison*



*Linear and Quadratic  
Discriminant Analysis  
with confidence  
ellipsoid*

Behind the scenes

# API implementation & contracts

# Method Chaining

Fit must return self:

```
X_scaled = StandardScaler().fit(X).transform(X)
```

```
pred = SVC().fit(X_train, y_train).score(  
    X_test, y_test)
```



# fit resets

```
from sklearn.tree import DecisionTreeClassifier
tree = DecisionTreeClassifier(max_depth=10)
tree.fit(iris.data, iris.target)
score_iris = tree.score(iris.data, iris.target)
tree.fit(digits.data, digits.target)
score_digits = tree.score(digits.data, digits.target)
```

# `__init__` does nothing

```
class PCA():  
    def __init__(self, n_components=None, copy=True, whiten=False,  
                  svd_solver='auto', tol=0.0, iterated_power='auto',  
                  random_state=None):  
        self.n_components = n_components  
        self.copy = copy  
        self.whiten = whiten  
        self.svd_solver = svd_solver  
        self.tol = tol  
        self.iterated_power = iterated_power  
        self.random_state = random_state
```

# get\_params & set\_params

```
def get_params(self, deep=True):
    out = dict()
    for key in signature(self.__init__):
        value = getattr(self, key, None)
        if deep and hasattr(value, 'get_params'):
            deep_items = value.get_params().items()
            out.update((key + '__' + k, val) for k, val in deep_items)
        out[key] = value
    return out
```

Used in GridSearchCV etc.

Not `__init__` !!!

All parameter validation etc in fit!

# (only) fit mutates self

- `__init__` only remembers construction parameters
- Transform / score etc don't change object
- Fit returns self but mutates object!

# Estimated attributes

## Attributes

-----

`components_` : array, shape (n\_components, n\_features)

Principal axes in feature space, representing the directions of maximum variance in the data. The components are sorted by ``explained\_variance``.

`explained_variance_` : array, shape (n\_components,)

The amount of variance explained by each of the selected components.

Equal to n\_components largest eigenvalues of the covariance matrix of X.

.. versionadded:: 0.18

`explained_variance_ratio_` : array, shape (n\_components,)

Percentage of variance explained by each of the selected components.

If ``n\_components`` is not set then all components are stored and the sum of the ratios is equal to 1.0.

`singular_values_` : array, shape (n\_components,)

The singular values corresponding to each of the selected components. The singular values are equal to the 2-norms of the ``n\_components`` variables in the lower-dimensional space.

Leaves two kinds of attributes:  
arguments to `__init__`; things estimated during fit

# fit\_transform / fit\_predict

In general: computational shortcut:

```
pca = PCA()  
pca.fit(X)  
X_pca = pca.transform(X)  
X_pca2 = pca.fit_transform(X)
```

Clustering / Manifold learning: Not inductive.

```
tsne = TSNE()  
X_tsne = tsne.fit_transform(X)
```

```
dbscan = DBSCAN()  
cluster_labels = dbscan.fit_predict(X)
```

# check\_estimator

```
class TemplateClassifier(BaseEstimator, ClassifierMixin):

    def __init__(self, demo_param='demo'):
        self.demo_param = demo_param

    def fit(self, X, y):

        # Check that X and y have correct shape
        X, y = check_X_y(X, y)
        # Store the classes seen during fit
        self.classes_ = unique_labels(y)

        self.X_ = X
        self.y_ = y
        # Return the classifier
        return self

    def predict(self, X):

        closest = np.argmin(euclidean_distances(X, self.X_), axis=1)
        return self.y_[closest]
```

`check_estimator(TemplateClassifier)`

AssertionError: Error message does not include the expected string: 'fit';  
Observed error message: "'TemplateClassifier' object has no attribute 'X\_'"

## Other API Building Blocks



# Scorers

```
grid = GridSearchCV(SVC(), params, scoring='roc_auc')
```

```
def my_acc_scorer(est, X, y):  
    y_pred = est.predict(X)  
    return (y == y_pred).mean()
```

```
grid = GridSearchCV(SVC(), params,  
                    scoring=my_acc_scorer)
```

# Cross-validation iterators

```
grid = GridSearchCV(SVC(), param_grid, cv=5)
```

```
cv = KFold(n_split=5, shuffle=True, random_state=3)
```

```
grid = GridSearchCV(SVC(), param_grid, cv=cv)
```

```
cv = RepeatedKFold(n_split=5, n_repeats=10)
```

```
grid = GridSearchCV(SVC(), param_grid, cv=cv)
```

# CV Iterators API

```
def split(self, X, y=None, groups=None):  
    """Generate indices to split data into training and test set.  
  
    Parameters  
    -----  
    X : array-like, shape (n_samples, n_features)  
        Training data, where n_samples is the number of samples  
        and n_features is the number of features.  
  
    y : array-like, of length n_samples  
        The target variable for supervised learning problems.  
  
    groups : array-like, with shape (n_samples,), optional  
        Group labels for the samples used while splitting the dataset into  
        train/test set.  
  
    Returns  
    -----  
    train : ndarray  
        The training set indices for that split.  
  
    test : ndarray  
        The testing set indices for that split.  
    """
```

```
kfold = KFold(n_splits=5)  
for train, test in kfold.split(X, y):  
    X_train = X[train]  
    X_test = X[test]
```

# Misc functions

- Scoring functions  
func(y\_true, y\_pred/y\_scores, ...)
- Pairwise Distances/kernels  
euclidean\_distances  
rbf\_kernel
- Evaluation:  
cross\_validate  
learning\_curve  
roc\_curve
- Helpers:  
train\_test\_split

# Development Practices

# Standards for OSS

Everything discussed in the open.  
Every convention and process documented.

# Development guide

<http://scikit-learn.org/dev/developers/contributing.html>

Contains:

- API details
- Bug report guidelines
- PR guidelines
- Reviewing guidelines
- How to find issues
- Details of CI

Even more at <http://scikit-learn.org/dev/developers/>

# Sphinx / numpydoc

```
class MultinomialNB(BaseDiscreteNB):
    """
    Naive Bayes classifier for multinomial models

    The multinomial Naive Bayes classifier is suitable for classification with
    discrete features (e.g., word counts for text classification). The
    multinomial distribution normally requires integer feature counts. However,
    in practice, fractional counts such as tf-idf may also work.

    Read more in the :ref:`User Guide <multinomial_naive_bayes>`.

    Parameters
    -----
    alpha : float, optional (default=1.0)
        Additive (Laplace/Lidstone) smoothing parameter
        (0 for no smoothing).

    fit_prior : boolean, optional (default=True)
        Whether to learn class prior probabilities or not.
        If false, a uniform prior will be used.

    class_prior : array-like, size (n_classes,), optional (default=None)
        Prior probabilities of the classes. If specified the priors are not
        adjusted according to the data.

    Attributes
    -----
    class_log_prior_ : array, shape (n_classes,)
        Smoothed empirical log probability for each class.

    intercept_ : property
        Mirrors ``class_log_prior_`` for interpreting MultinomialNB
        as a linear model.

    feature_log_prob_ : array, shape (n_classes, n_features)
        Empirical log probability of features
        given a class, ``P(x_i|y)``.

    coef_ : property
        Mirrors ``feature_log_prob_`` for interpreting MultinomialNB
        as a linear model.

    class_count_ : array, shape (n_classes,)
        Number of samples encountered for each class during fitting. This
        value is weighted by the sample weight when provided.

    feature_count_ : array, shape (n_classes, n_features)
        Number of samples encountered for each (class, feature)
        during fitting. This value is weighted by the sample weight when
        provided.

    Examples
    -----
    >>> import numpy as np
    >>> X = np.random.randint(5, size=(6, 100))
    >>> y = np.array([1, 2, 3, 4, 5, 6])
    >>> from sklearn.naive_bayes import MultinomialNB
    >>> clf = MultinomialNB()
    >>> clf.fit(X, y)
    MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
    >>> print(clf.predict(X[2:3]))
    [3]

    Notes
    -----
    For the rationale behind the names ``coef`` and ``intercept``, i.e.
    naive Bayes as a linear classifier, see J. Rennie et al. (2003),
    Tackling the poor assumptions of naive Bayes text classifiers, ICML.

    References
    -----
    C.D. Manning, P. Raghavan and H. Schuetze (2008). Introduction to
    Information Retrieval. Cambridge University Press, pp. 234-265.
    http://nlp.stanford.edu/IR-book/html/htmledition/naive-bayes-text-classification-1.html
    """
```

## sklearn.naive\_bayes.MultinomialNB

```
class sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)
```

[\[source\]](#)

Naive Bayes classifier for multinomial models

The multinomial Naive Bayes classifier is suitable for classification with discrete features (e.g., word counts for text classification). The multinomial distribution normally requires integer feature counts. However, in practice, fractional counts such as tf-idf may also work.

Read more in the User Guide.

**Parameters:** `alpha` : float, optional (default=1.0)

Additive (Laplace/Lidstone) smoothing parameter (0 for no smoothing).

`fit_prior` : boolean, optional (default=True)

Whether to learn class prior probabilities or not. If false, a uniform prior will be used.

`class_prior` : array-like, size (n\_classes), optional (default=None)

Prior probabilities of the classes. If specified the priors are not adjusted according to the data.

**Attributes:** `class_log_prior_` : array, shape (n\_classes,)

Smoothed empirical log probability for each class.

`intercept_` : array, shape (n\_classes,)

Mirrors `class_log_prior_` for interpreting MultinomialNB as a linear model.

`feature_log_prob_` : array, shape (n\_classes, n\_features)

Empirical log probability of features given a class,  $P(x_i|y)$ .

`coef_` : array, shape (n\_classes, n\_features)

Mirrors `feature_log_prob_` for interpreting MultinomialNB as a linear model.

`class_count_` : array, shape (n\_classes,)

Number of samples encountered for each class during fitting. This value is weighted by the sample weight when provided.

`feature_count_` : array, shape (n\_classes, n\_features)

Number of samples encountered for each (class, feature) during fitting. This value is weighted by the sample weight when provided.

### Notes

For the rationale behind the names `coef_` and `intercept_`, i.e. naive Bayes as a linear classifier, see J. Rennie et al. (2003), Tackling the poor assumptions of naive Bayes text classifiers, ICML.

### References

C.D. Manning, P. Raghavan and H. Schuetze (2008). Introduction to Information Retrieval. Cambridge University Press, pp. 234-265. <http://nlp.stanford.edu/IR-book/html/htmledition/naive-bayes-text-classification-1.html>

### Examples

```
>>> import numpy as np
>>> X = np.random.randint(5, size=(6, 100))
>>> y = np.array([1, 2, 3, 4, 5, 6])
>>> from sklearn.naive_bayes import MultinomialNB
>>> clf = MultinomialNB()
>>> clf.fit(X, y)
MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
>>> print(clf.predict(X[2:3]))
[3]
```

### Methods

<code>fit(X, y[, sample_weight])</code>	Fit Naive Bayes classifier according to X, y
<code>get_params([deep])</code>	Get parameters for this estimator.
<code>partial_fit(X, y[, classes, sample_weight])</code>	Incremental fit on a batch of samples.
<code>predict(X)</code>	Perform classification on an array of test vectors X.
<code>predict_log_proba(X)</code>	Return log-probability estimates for the test vector X.
<code>predict_proba(X)</code>	Return probability estimates for the test vector X.
<code>score(X, y[, sample_weight])</code>	Returns the mean accuracy on the given test data and labels.
<code>set_params(**params)</code>	Set the parameters of this estimator.



# Sphinx / numpydoc

```
.. _svm:
```

## Support Vector Machines

```
.. currentmodule:: sklearn.svm
```

**Support vector machines (SVMs)** are a set of supervised learning methods used for :ref:`classification <svm\_classification>`, :ref:`regression <svm\_regression>` and :ref:`outliers detection <svm\_outlier\_detection>`.


The advantages of support vector machines are:

- Effective in high dimensional spaces.
- Still effective in cases where number of dimensions is greater than the number of samples.
- Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.
- Versatile: different :ref:`svm kernels` can be specified for the decision function. Common kernels are provided, but it is also possible to specify custom kernels.

The disadvantages of support vector machines include:

- If the number of features is much greater than the number of samples, the method is likely to give poor performances.
- SVMs do not directly provide probability estimates, these are calculated using an expensive five-fold cross-validation (see :ref:`Scores and probabilities <scores\_probabilities>`, below).

The support vector machines in scikit-learn support both dense (`numpy.ndarray` and convertible to that by `numpy.asarray`) and sparse (any `scipy.sparse`) sample vectors as input. However, to use an SVM to make predictions for sparse data, it must have been fit on such data. For optimal performance, use C-ordered `numpy.ndarray` (dense) or `scipy.sparse.csr_matrix` (sparse) with `dtype=float64`.



[Home](#) [Installation](#) [Documentation](#) [Examples](#)

Previous  
1.3. Kernel ...

Next  
1.5. Stochastic ...

Up  
1. Supervised ...

This documentation is for  
scikit-learn **version**  
**0.18.1** — [Other versions](#)

If you use the software,  
please consider citing  
[scikit-learn](#).

### 1.4. Support Vector Machines

1.4.1. Classification

- 1.4.1.1. Multi-class classification
- 1.4.1.2. Scores and probabilities
- 1.4.1.3. Unbalanced problems

1.4.2. Regression

1.4.3. Density estimation, novelty detection

1.4.4. Complexity

1.4.5. Tips on Practical Use

1.4.6. Kernel functions

- 1.4.6.1. Custom Kernels
  - 1.4.6.1.1. Using Python functions as kernels
  - 1.4.6.1.2. Using the Gram matrix
  - 1.4.6.1.3. Parameters of the RBF Kernel

1.4.7. Mathematical formulation

## 1.4. Support Vector Machines

**Support vector machines (SVMs)** are a set of supervised learning methods used for [classification](#), [regression](#) and [outliers detection](#).

The advantages of support vector machines are:

- Effective in high dimensional spaces.
- Still effective in cases where number of dimensions is greater than the number of samples.
- Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.
- Versatile: different [Kernel functions](#) can be specified for the decision function. Common kernels are provided, but it is also possible to specify custom kernels.

The disadvantages of support vector machines include:

- If the number of features is much greater than the number of samples, the method is likely to give poor performances.
- SVMs do not directly provide probability estimates, these are calculated using an expensive five-fold cross-validation (see [Scores and probabilities](#), below).

The support vector machines in scikit-learn support both dense (`numpy.ndarray` and convertible to that by `numpy.asarray`) and sparse (any `scipy.sparse`) sample vectors as input. However, to use an SVM to make predictions for sparse data, it must have been fit on such data. For optimal performance, use C-ordered `numpy.ndarray` (dense) or `scipy.sparse.csr_matrix` (sparse) with `dtype=float64`.

# Deprecations / backward compatibility

- Don't change any behavior (except bug fixes)


```
@property
@deprecated("Attribute labels_ was deprecated in version 0.13 and "
            "will be removed in 0.15. Use 'classes_' instead")
def labels_(self):
    return self.classes_
```

# Practical Notes on Contributing

# Describing PR

## Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#).

 base fork: **scikit-learn/scikit-learn** ▼

base: **master** ▼


...

head fork: **amueller/scikit-learn** ▼

compare: **my\_feature\_branch** ▼

✓ **Able to merge.** These branches can be merged.

Please review the [guidelines for contributing](#) to this repository.



[MRG] minor pep8

Write

Preview

AA ▼ B i “ < > 🔗

⋮ ⋮ ⋮

↶ @ 🚩

<!--

Thanks for contributing a pull request! Please ensure you have taken a look at the contribution guidelines: <https://github.com/scikit-learn/scikit-learn/blob/master/CONTRIBUTING.md#Contributing-Pull-Requests>

-->

### Reference Issue

<!-- Example: Fixes #1234 -->

Attach files by dragging & dropping, [selecting them](#), or pasting from the clipboard.

☒ **Allow edits from maintainers.** [Learn more](#)

Create pull request

Reviewers

No reviews—request one

Assignees

No one—assign yourself

Labels

None yet

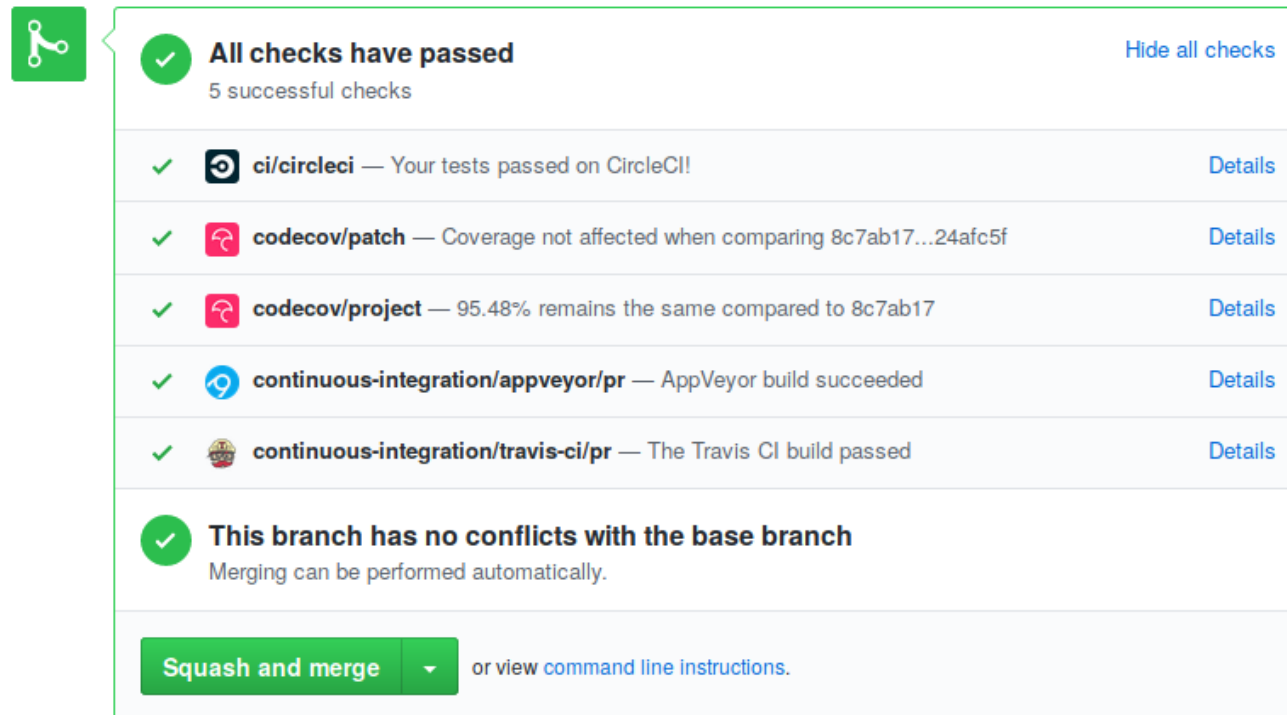
Milestone

No milestone



[WIP] = work in progress  
[MRG] = ready to merge











# Regression tests


- Are mandatory! For everything (except documentation changes)!
- Make sure continuous integration passes:



The screenshot shows the GitHub Actions status bar for a pull request. It features a green checkmark icon on the left. The main status is "All checks have passed" with a link to "Hide all checks". Below this, a list of five checks is shown, each with a green checkmark, an icon, a name, a description, and a "Details" link. The checks are: "ci/circleci" (Your tests passed on CircleCI), "codecov/patch" (Coverage not affected when comparing 8c7ab17...24afc5f), "codecov/project" (95.48% remains the same compared to 8c7ab17), "continuous-integration/appveyor/pr" (AppVeyor build succeeded), and "continuous-integration/travis-ci/pr" (The Travis CI build passed). At the bottom, there is a green "Squash and merge" button with a dropdown arrow, followed by the text "or view command line instructions."

  **All checks have passed** [Hide all checks](#)  
5 successful checks

	 <b>ci/circleci</b> — Your tests passed on CircleCI <a href="#">Details</a>
	 <b>codecov/patch</b> — Coverage not affected when comparing 8c7ab17...24afc5f <a href="#">Details</a>
	 <b>codecov/project</b> — 95.48% remains the same compared to 8c7ab17 <a href="#">Details</a>
	 <b>continuous-integration/appveyor/pr</b> — AppVeyor build succeeded <a href="#">Details</a>
	 <b>continuous-integration/travis-ci/pr</b> — The Travis CI build passed <a href="#">Details</a>

 **This branch has no conflicts with the base branch**  
Merging can be performed automatically.

[Squash and merge](#) or view [command line instructions](#).

# What's next?

- Wait for reviews (be patient).
- Address review comments in the same branch.
- Pushing to your fork will update the PR
- Reviewers will “approve” PR or change title to [MRG + 1]
- You need two approvals for a merge.

# Finding Issues

- Check “need contributor”, “easy” and “sprint” issues.
- Something unclear in the docs? Fix it!
- Can’t fix something that’s unclear? Open an issue!
- Problem that you keep running into: Open an issue!
- Find stalled PRs (the author didn’t address reviews for  $\sim > 1$  month) and continue them!

# Reviewing

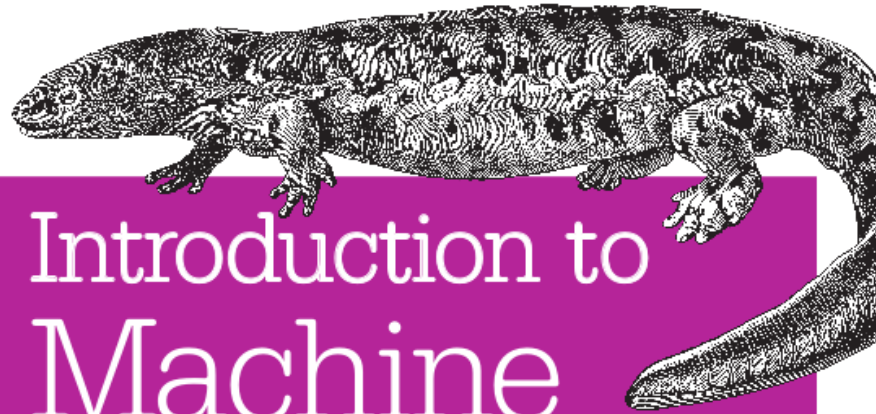
- You can review PRs and issues!
- Some bugs are not confirmed. See if you can confirm them and under what conditions?
- You can review documentation PRs for language and whether they are clear to you.
- You can review code changes on whether they address the issue (might be a bit tricky).
- Don't be afraid to ask clarifying questions!



# Final words

- Pick something TRIVIALY SIMPLE as the first contribution.
- You can do the cool stuff afterwards!
- There might be interesting issues that are not appropriately tagged.
- Consider finishing up stalled PRs

O'REILLY®



# Introduction to Machine Learning with Python

A GUIDE FOR DATA SCIENTISTS

Andreas C. Müller & Sarah Guido

# Thank you.

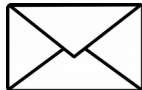


@amuellermi



@amueller

amueller@nyu.edu



<http://amueller.io>

